2008-04-01

A Semantic Based Search Engine for Open Architecture Requirements Documents

Craig Martell

A Semantic-Based Search Engine for Open Architecture Requirements Documents

Dr. Craig Martell

Associate Professor, Naval Postgraduate School

Our Team

- Paige Adams, LT USN
- Pranav Anand, Assistant Prof, Linguistics, UCSC
- Grant Gehrke, ENS USN
- Ralucca Gera, Assistant Prof, Mathematics, NPS
- Marco Draeger, CPT German Army
- Craig Martell, Associate Prof, Computer Science, NPS
- Kevin Squre, Assistant Prof, Computer Science, NPS

The ReSEARCH Project: What we're up to.

- Using open-sourced components, we want to design a semantic search engine for requirements documents that supports the SHARE repository
- We need to match over the meaning of a *requirement*, not a question or a query string.
- Do processing to "enrich" both the query and the documents with semantic information.
- Automatically augment ontologies with new hypernomy ("is a") and mereology ("is a part of") relations.
  - That is, do we have to be told that a Hummer is a vehicle, and one of its parts is a steering wheel, or can we discover this from the text.
- Etc.

Why use semantic search?

- Existing keyword-based search engines do not take into account the semantics of the documents they are searching.

- This is important when trying to find components that do what you need, not what you type.

Why use semantic search?

- The query string and the desired documents may not use the same phrases.

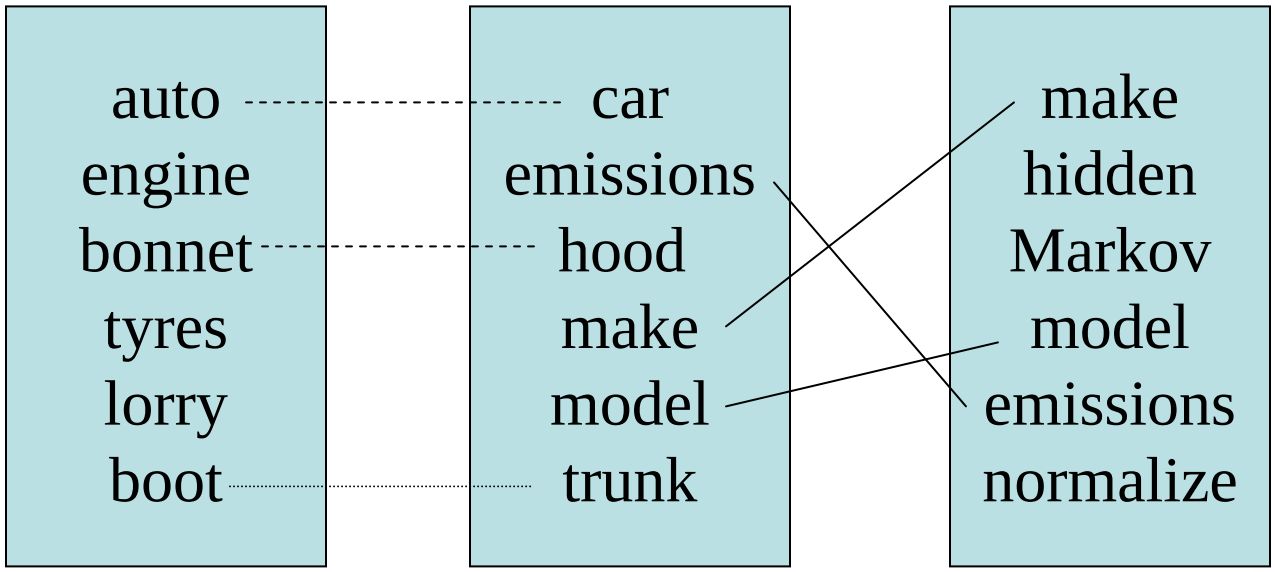
  Q: What fuel does the F-22A consume?

  A: *The F-22A's Raptor uses JP-8.*

- Query and answer convey same meaning, but use different forms
  - Here, "consume" and "uses" are *synonymous.*

Prior and current strategies

Keyword Search Model



Synonymy Problem

Polysemy Problem

Prior and current strategies

- Brin and Page (1998) revolutionized search by using PageRank, which changes the order in which the pages that match the keywords in the query are returned.

- The essence of the Google innovation is in how the PageRank algorithm works.

PageRank algorithm

The rank of a particular page depends on:

- The number of pages pointing to it,
- The rank of each page pointing to it,
- The number of outgoing links on each those pages.

PageRank algorithm

- **PageRank metric** PR(P) defines recursively the rank/importance of each page P by

$$PR(P) \propto \frac{PR(T_1)}{C(T_1)} + \frac{PR(T_2)}{C(T_2)} + \ldots + \frac{PR(T_n)}{C(T_n)}$$

where

- $T_1, T_2, \ldots$ are all the pages pointing to P
- each T_i has $C(T_i)$ outgoing links.

Random Surfer

To further determine the rank of all web pages
Google simulates the behavior of virtual surfers randomly surfing the web.

A page's rank is then updated based on how frequently the random surfers visit that page.

This pre-existing rank of each individual website is assigned independently of any query.

Expert Rank

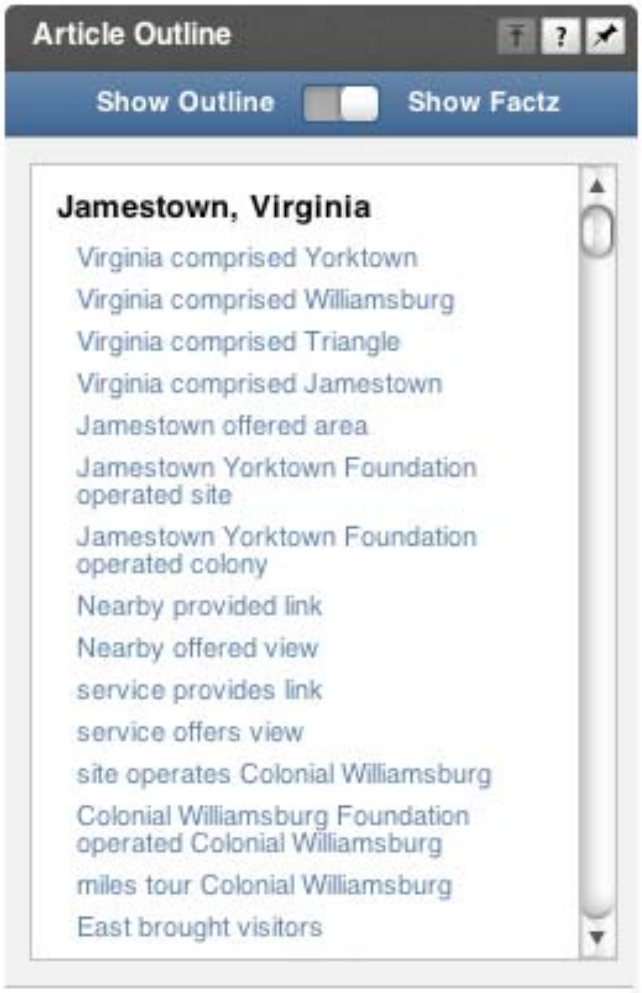
Ask.com (Ask Jeeves) uses the ExpertRank algorithm:

- uses the number of incoming links as well
- attempts to identify topic clusters related to search
- find experts within these topics to “seed” the rank of some websites as “expert” sites.
- PageRank has the problem that “correct” is not the same as “highly-ranked.”

Current Online Semantic Search

- Powerset Labs has emerged as a forerunner in online semantic search using natural language to extract facts from text.
- On 11 May 08, Powerset's search moved from beta to a public release
- Currently, Powerset searches only Wikipedia documents, but intends to expand search to the Internet in the future.

Powerset Indexing System



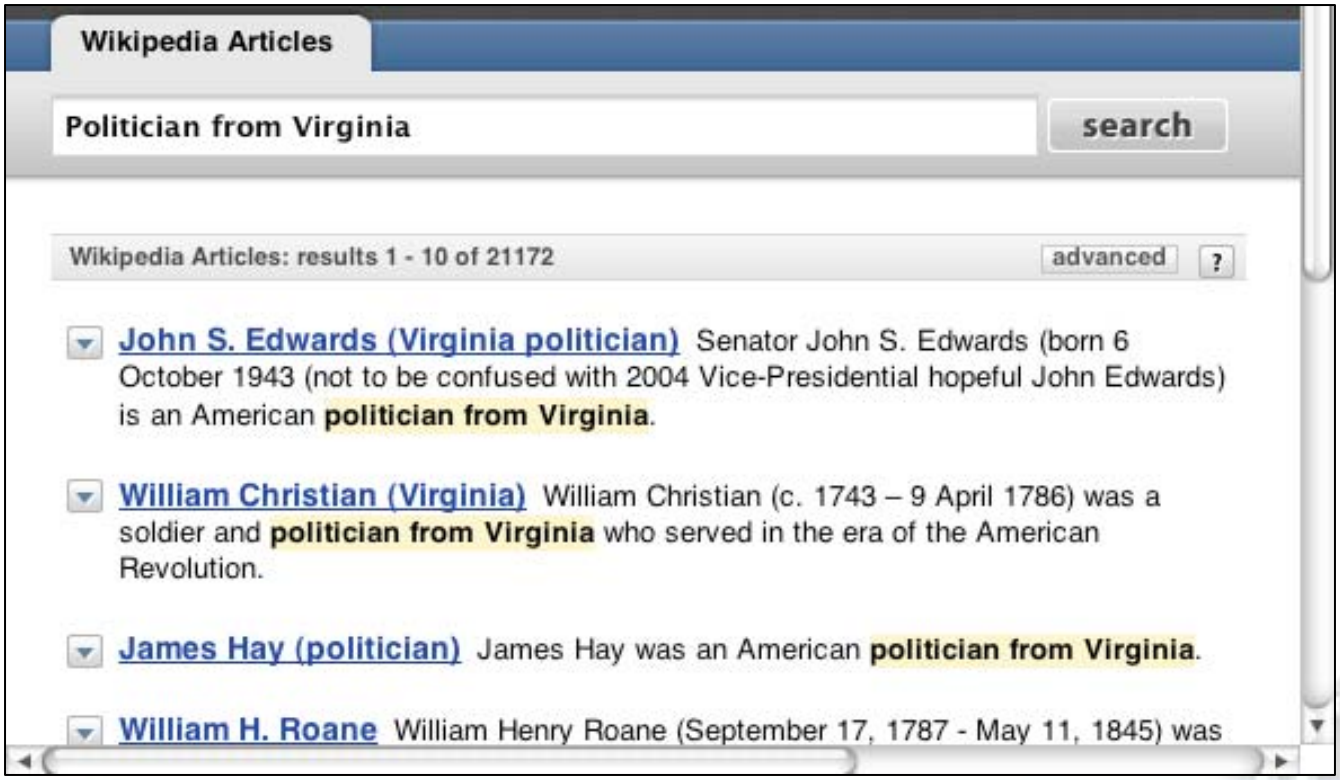
- Powerset's algorithms are not publicly available, but their behavior can be inferred from publicly available demos
  - Powerset parses documents to extract "factz"
  - "Factz" are generally triples of subject-verb-object
  - Search is performed over these "factz" rather than the full text

Question Answering

- Keywords such as “When” tell the system how to narrow results
- “W” words such as “Who” and “When” act as wildcards for matching “factz,” allowing many searches to be matched exactly

Question Answering

- Other functional words such as “From” in the search “Politician from Virginia” improve results significantly over searching on just the keywords “Politician” and “Virginia”

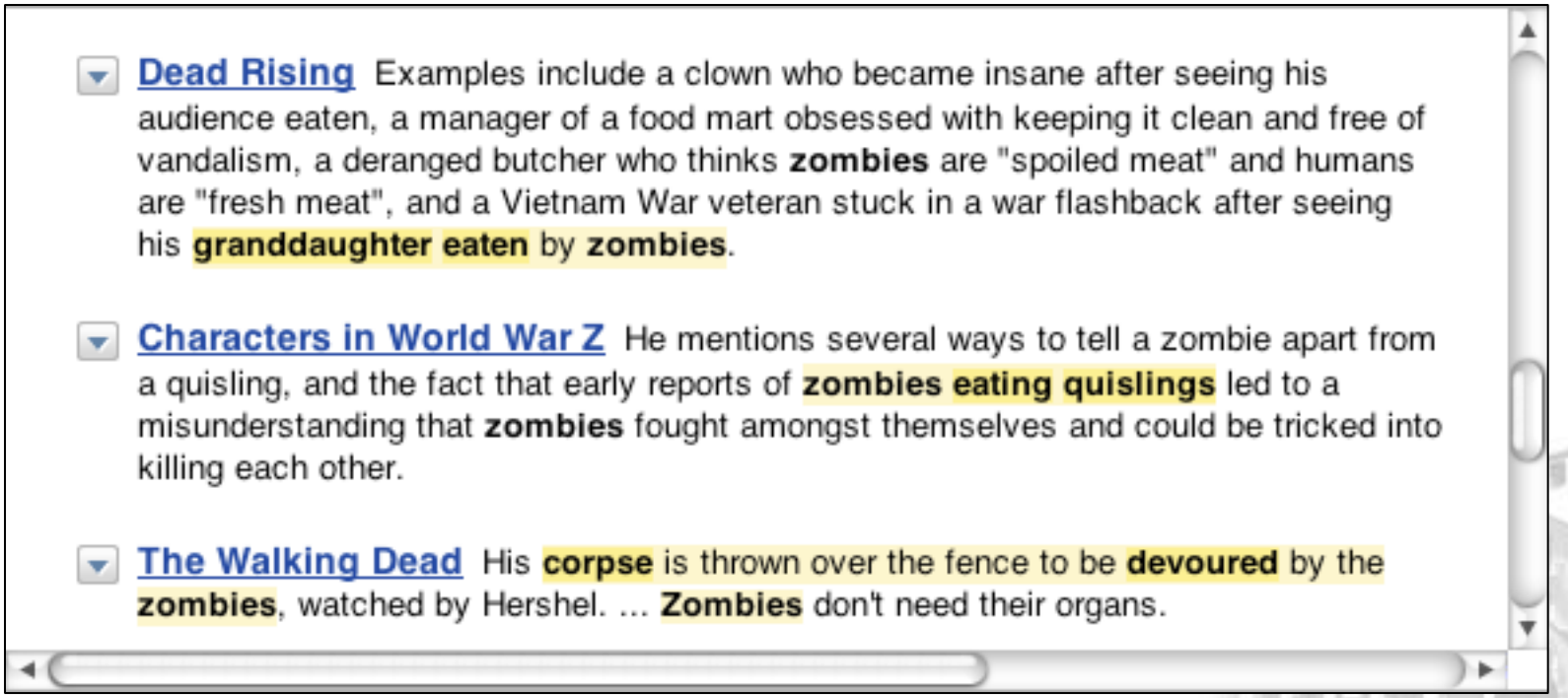


Question Answering

- Question Answering task does not align exactly with requirements document search
- Requirements documents do not hold “Answers” to questions
- Encoding of facts is, however, useful
  - Computationally less demanding
  - Efficient use of storage space for the index
  - Allows *domain specific constructs* of facts to be formulated and recognized in the corpus

Query Expansion with Synonymy

- The search “What do zombies eat?” suggests that Powerset searches for synonyms of query terms, matching “devour”
- Additionally, stemming matches the inverted form “eaten by”



Query Expansion with Synonymy

- Stemming of terms is found in most search engines and is fairly easy to perform
- Matching synonyms allows “close” matches on meaning without requiring an exact keyword match
- Using a structured ontology, expansion is not limited to synonyms but may be extended to hypernyms, hyponyms, and meronyms as well
  - Ontology based query expansion does not appear to be used in current Powerset searches
  - One of our primary approaches:
    - Research Question: Can we automatically augment a given ontology using the text of the documents?

Discovering Synonymous Sentences

- Harris (1954): Synonymous words will occur in the same kinds of environments
- Lin & Pantel (2001): Synonymous sentences will contain the same kinds of words

The F-22A consumes JP-8

The F-22A's engine uses JP-8

- Idea: construct sentence similarity metric

Discovering Synonymous Sentences

- Sentence similarity is the geometric average of the similarity of the **positions** in the sentence:

$$\text{sim}(X_1 \text{ consumes } Y_1, X_2\text{'s engine uses } Y_2) = \sqrt{sim(X_1, X_2) \times sim(Y_1, Y_2)}$$

$$\text{sim}(X_1 \quad p_1 \quad Y_1, X_2 \quad p_2 \quad Y_2) = \sqrt{sim(X_1, X_2) \times sim(Y_1, Y_2)}$$

Discovering Synonymous Sentences

- Position similarity is a normalized sum of the pointwise mutual information of all words that appear in both positions of the respective paths:

$$sim(X_1, X_2) = \frac{\sum_{w \in T(p_1,s) \cap T(p_2,s)} (mi(p_1, s, w) + mi(p_2, s, w))}{\sum_{w \in T(p_1,s)} mi(p_1, s, w) + \sum_{w \in T(p_2,s)} mi(p_2, s, w)}$$

$$mi(X_1 \ldots p \ldots Y_1, X_1, w) = \log \frac{\frac{f(p, X_1 = w)}{f(*, X_1 = w)}}{\frac{f(p, X_1 = *)}{f(*, X_1 = w)}}$$

Discovering Synonymous Sentences

- Lin & Pantel evaluated system against TREC-8 Question Answering Task question set.

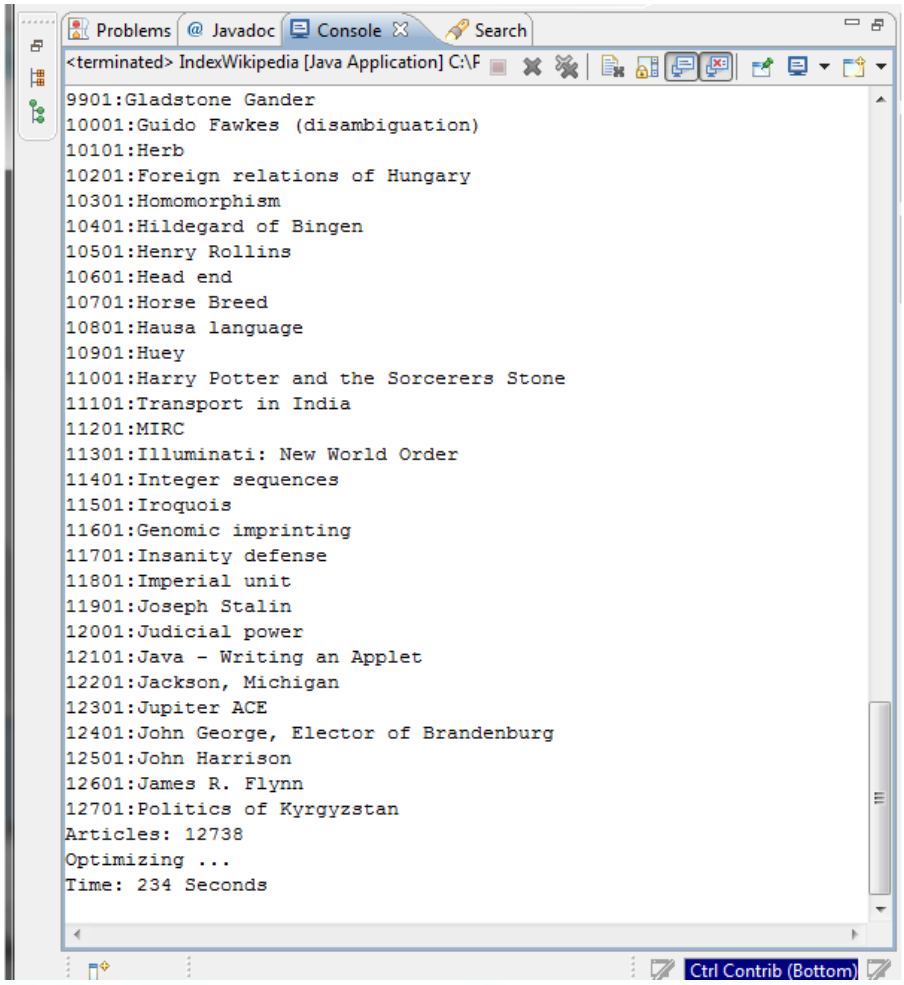
Query	# Paths	Accuracy
X is author of Y	21	52.5%
X is monetary value of Y	0	N/A
X manufactures Y	37	92.5%
X spend Y	16	40.0%
spend X on Y	15	37.5%
X is managing director of Y	14	35.0%
X asks Y	23	57.5%
asks X for Y	14	35.0%
X asks for Y	21	52.5%

The ReSEARCH Project: Work for us.

- Using open-sourced components, design a semantic search engine for requirements documents that supports the SHARE repository
- Match over the meaning of a *requirement*, not a question.
- Do processing to “enrich” both the query and the documents with semantic information.
- Automatically augment ontologies with new hypernomy (“is a”) and mereology (“is a part of”) relations.
  - That is, do we have to be told that a Hummer is a vehicle, and one of its parts is a steering wheel, or can we discover this from the text.
- Lots more!

Backup Slides

Building the Index



A Sample Search with Lucene

```
>>> def searchWikipedia(queryString):
        query = parser.parse(queryString)
        hits = searcher.search(query)
        print "Hits: ",hits.length()
        for i in range(0, hits.length()):
                doc = hits.doc(i)
                title = doc.get("title")
                print i,": ",title, "score: ", hits.score(i)


>>> searchWikipedia("scream AND munch")
Hits:  6
0 :  Edvard Munch score:  0.999999940395
1 :  Afterglow score:  0.4743026793
2 :  Angst score:  0.23715133965
3 :  Fear score:  0.142290815711
4 :  August 31 score:  0.118575669825
5 :  August 22 score:  0.117710016668
>>>
```

Using an Augmented Search String

```
>>> searchWikipedia("Relativity")
Hits:  106
0 :  General Relativity score:  1.0
>>> searchWikipedia("text:relativity text:einstein")
Hits:  206
0 :  Albert Einstein score:  1.0
1 :  Inertial frame of reference score:  0.812765300274
2 :  Gravitational redshift score:  0.801645994186
3 :  General Relativity score:  0.787778377533
4 :  General relativity score:  0.78440785408
5 :  Acceleration score:  0.636109173298
6 :  Arthur Stanley Eddington score:  0.603332340717
7 :  Cosmic censorship hypothesis score:  0.578752875328
8 :  Graviton score:  0.577827572823
9 :  Einstein score:  0.574990808964
10 :  Faster-than-light score:  0.571875691414
>>>
```

Our Work

GOAL: Design an alternative method to explicitly store/represent semantic metadata in order to enable semantic search.